

Controlling the weights of simulation particles: adaptive particle management using k -d trees

Jannis Teunissen^{a,*}, Ute Ebert^{a,b}

^a*Centre for Mathematics and Informatics (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

^b*Department of Physics, Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands*

Abstract

In particle simulations, the weights of particles determine how many physical particles they represent. Adaptively adjusting these weights can greatly improve the efficiency of the simulation, without creating severe nonphysical artifacts. We present a new method for the pairwise merging of particles. Pairwise merging reduces the number of particles by combining two particles into one. To find particles that are ‘close’ to each other, we use a k -d tree data structure. With a k -d tree, close neighbors can be searched for efficiently, and independently of the mesh used in the simulation. The merging can be done in different ways, conserving for example momentum or energy. We introduce probabilistic schemes, which set properties for the merged particle using random numbers. The effect of various merge schemes on the energy distribution, the momentum distribution and the grid moments is compared.

Keywords: super-particle, macro-particle, adaptive particle management, coalescence, k -d tree, particle simulations, particle in cell

1. Introduction

Particle-based simulations are widely used, for example to study fluid flows or plasmas. The physical particles of interest are often not simulated individually, but as groups of particles, called *super-particles* or *macro-particles*. Most systems contain so many particles that simulating them individually would be very slow or impossible. And for many macroscopic properties of a system, individual particle behavior is not important. On the other hand, a sufficient number of particles is required to limit stochastic fluctuations.

The weight of a simulation particle indicates how many physical particles it represents. Traditionally, particles had a fixed weight [1, 2]. More recently, Lapenta and Brackbill [3–5], Assous et al. [6], Welch et al. [7] and others have introduced methods that adapt the weight of particles during a simulation. As discussed in [6], adaptive methods have significant advantages if:

1. Many new particles are created in the simulation. Adaptive re-weighting is required to limit the total number of particles. Examples can be found in [8] and [9].
2. The system has a multiscale nature. In some regions more macro-particles are required, especially if some type of mesh refinement is employed, see for example [10].
3. Control is needed over the number of particles per cell, for example to limit stochastic noise to a realistic value.

The goal of these methods is to change the number of particles to a desired value, while keeping the distribution of particles intact. Most methods operate on a single grid cell at a time; arguments for this approach are given in [5]. There are different ways to change the number of particles. One option is to

*Corresponding author

Email address: Jannis.Teunissen@cwi.nl (Jannis Teunissen)

merge two (or sometimes three) particles, to form particles with higher weights. Reversely, splitting can be performed to reduce weights. Another option is to replace all the particles in a cell by a new set of particles, with different weights. We will use the name ‘adaptive particle management’, introduced in [7], for such algorithms.

We present a technique for the merging of particles, that extends earlier work of Lapenta [3]. This method can operate independently of the mesh, and in any space dimension. The main idea is to store the particle coordinates (typically position and velocity) in a k -d tree. A k -d tree is a space partitioning data structure that given N points enables searching for neighbors in $O(\log N)$ time [11]. We can then efficiently locate pairs of particles with similar coordinates, and these pairs can be merged. Because the merged particles are similar, the total distribution of particles is not significantly altered.

In section 2 we briefly discuss the general principles of particle management and k -d trees. The implementation of the new particle management algorithm is discussed in section 3. In section 4 we provide numerical examples to demonstrate the method, and we compare different ways of merging particles.

2. Adaptive particle management and k -d trees

As stated in the introduction, it is typically impossible to simulate all the physical particles in a system individually. Therefore super-particles are used, representing multiple physical particles. Often, the simulation can run faster or give more accurate results if the weight of these super-particles is controlled adaptively. Different names have been introduced for these algorithms: ‘adaptive particle management’ [7], ‘control of the number of particles’ [5], ‘particle coalescence’ [6], ‘particle resampling’ [8], ‘particle remapping’ [12], ‘particle rezoning’ [3], ‘(particle) number reduction method’ [13] and probably others. There seem to be many independent findings, with independent names. We will use the name ‘adaptive particle management’ (APM), introduced in [7], to describe this class of algorithms.

2.1. Conservation properties

If weights of particles are adjusted, then the ‘microscopic details’ of a simulation are changed. But the relevant macroscopic quantities should be conserved as much as possible. To specify these macroscopic quantities, we consider a very common type of particle simulation: the particle in cell (PIC) method, also known as the particle mesh (PM) method [1, 2]. In PIC simulations, particles are mapped to moments on a grid. From the grid moments the fields acting on the particles are computed, and the particles move accordingly. For example, in an electrostatic code, the charge density is used to compute the electric field.

An APM algorithm typically changes a set of N_{in} particles to a new set of N_{out} particles. If the two sets give rise to the same grid moments, they give rise to the same fields. Therefore, most algorithms are designed to (approximately) conserve the relevant grid moments.

Only conserving the grid moments is not enough, because the dynamics of a system are not fully determined by the fields. For example, the results of a simulation can be very sensitive to changes in the momentum or energy distribution. Therefore, some methods try to preserve the shape of these distributions. More generally, we would like to keep the important aspects of the particle distribution function $f(\mathbf{x}, \mathbf{v}, t)$ the same. The changes to $f(\mathbf{x}, \mathbf{v}, t)$ should not be significantly larger than the fluctuations that naturally occur. For example, in a collision dominated plasma, particles frequently change direction. Not conserving the momentum distribution in each direction might have little effect on the overall results. But for a collisionless plasma, a change in the momentum distribution might lead to significant differences. Similarly, due to the finite number of particles, fluctuations in the local particle density occur naturally. Therefore, keeping the particle density exactly the same on each grid point might not be necessary, as long as the total number of particles is conserved.

2.2. Merging and splitting particles

A set of N_{in} particles can be transformed to a new set of N_{out} particles in many ways. If $N_{\text{in}} > N_{\text{out}}$, we use a pairwise coalescence algorithm, that merges two particles into a single new one. Compared to algorithms that transform multiple particles at the same time, pairwise coalescence has two advantages.

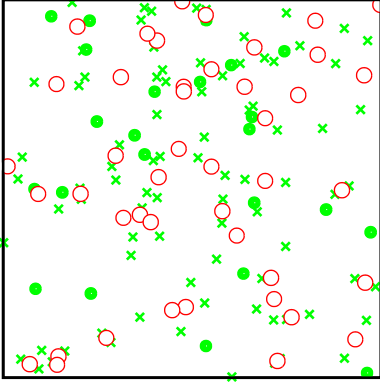


Figure 1: Example showing the merging of particles close in space and velocity (velocity is not shown). The particles that were removed after merging are shown as green crosses, particles that were not merged as green filled circles, and the newly formed merged particles as red empty circles. The latter have weight 2, the rest weight 1.

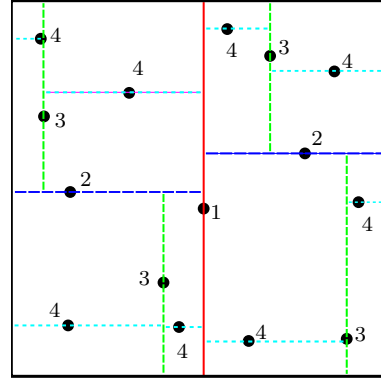


Figure 2: Schematic example of how a k -d tree is generated for points in the plane (indicated as black dots). At every step (indicated by the numbers), boxes are split in two parts. The split is located on a point, that is added to the tree. The direction of splitting alternates between vertical and horizontal.

First, it is a more local operation, because only closest neighbors in phase space are selected. This ensures that the distribution of particles is not changed very much. Second, it involves fewer degrees of freedom, which makes it simpler to set the properties for new particles. The pairwise coalescence of particles is illustrated in figure 1.

In D dimensions, the momentum \mathbf{p} of the new particle has D degrees of freedom. Imposing momentum and energy conservation puts $D + 1$ constraints on \mathbf{p} . Therefore, it is in general not possible to conserve both energy and momentum in pairwise coalescence. This means that there is no single best way to merge particles, as different applications require the conservation of different properties. We consider several coalescence schemes, which are discussed in section 3.2.

The situation would be very different if N_{in} particles are merged at the same time to form multiple new particles. We still have $D + 1$ constraints, but now $D \cdot N_{\text{out}}$ degrees of freedom in the momenta of the N_{out} new particles. The system is under-determined, and additional information about the particles has to be used. This leads to more complicated algorithms, see for example [6, 7].

If $N_{\text{in}} < N_{\text{out}}$, particles have to be split. Several methods for particle splitting have been compared by Lapenta in [3]. As shown there, choosing the right splitting method can be important, depending on the type of simulation. Here, we will not consider this problem in detail, as our focus is on the merging of particles. We will simply split single particles into two new ones with the same properties, but half the weight. This can be viable if the simulation includes random collisions, so that the new particles will undergo different collisions and spread out.

2.3. k -d trees

To locate particles with similar coordinates we use a k -d tree [11], which is a space partitioning data structure. A k -d tree can be used to organize a set of points in a k -dimensional space, for any $k \geq 1$. The tree consists of nodes, that contain data (the coordinates of one of the points) and links to at most two ‘child’-nodes. The starting point of the tree is the root node, and it contains as many nodes as there are points.

We will briefly explain how such a k -d tree can be generated. To help with the explanation, we let nodes have a **todo** list, that contains points that need to be processed. Suppose we have a collection of points in the (x, y) plane. Initially all points are in the **todo** list of the root node. Then the following algorithm, which is illustrated in figure 2, creates the k -d tree:

1. Pick a splitting coordinate, either x or y . A simple choice is to alternate between them.
2. For each node with a non-empty **todo** list:

- (a) Sort the particles in the list along the splitting coordinate. The particle in the middle of the list is the median. If the list contains an even number of particles, pick one of the two middle particles as the median.
 - (b) The point corresponding to the median is assigned to the node.
 - (c) The remaining points are moved to the **todo** lists of (at most) two new child nodes. The first one gets the points below the median, the second one those above the median.
3. If there are still points in **todo** lists, go back to step one. Otherwise, the tree is completed.

In k dimensions, the only difference would be that there are now k choices for the splitting coordinate. The computational complexity of creating a k -d tree like this is $O(N \log^2 N)$, with N the number of points in the tree. This can be reduced to $O(N \log N)$ if a linear-time median finding algorithm is used instead of sorting at step 2a. Searching for the nearest neighbor to a location \mathbf{r} can be done in $O(\log N)$ time. The basic idea is to first traverse the tree down from the root node, at each step selecting that side of the tree that \mathbf{r} lies in. (If \mathbf{r} happens to lie exactly on a splitting plane, it is a matter of convention which side to pick.) During the search, the closest neighbor found so far is stored. Then going upward in the tree, at every step determine whether a closer neighbor could lie on the other side of the splitting plane. If so, also traverse that other part of the tree down (but only where it can contain a closer neighbor). Typically, only a small number of these extra traverses is required. When the algorithm ends up at the root node again, the overall closest neighbor is found.

For the numerical tests presented in section 4, we have used the Fortran 90 version of the KDTree2 [14] library.

3. Implementation

We will discuss the implementation of our adaptive particle management algorithm in section 3.1. Different schemes that can be used for particle merging are given in section 3.2.

3.1. Adaptive particle management algorithm

Suppose that we have particles with coordinates \mathbf{x}_i , \mathbf{v}_i and weights w_i . Furthermore, assume there is some function $W_{\text{opt}}(i)$ that gives the user-determined optimal weight for particle i . Then the APM algorithm works as follows:

1. Create a list **merge** with all the particles for which $w_i < \frac{2}{3}W_{\text{opt}}(i)$. Similarly, create a list **split** with particles for which $w_i > \frac{3}{2}W_{\text{opt}}(i)$.

The function $W_{\text{opt}}(i)$ gives the desired weight for particle i . The factors $\frac{2}{3}$ and $\frac{3}{2}$ ensure that merged particles are not directly split again, and vice versa. A good choice of $W_{\text{opt}}(i)$ will often depend on the application. We typically want to keep the number of particles per cell close to a desired value N_{ppc} , and use $W_{\text{opt}}(i) = \max(1, N_{\text{phys}}(i)/N_{\text{ppc}})$. Here $N_{\text{phys}}(i)$ denotes the number of physical particles in the cell of particle i . This increases the number of particles in regions with finer grids.

2. For the particles in **merge**:

- (a) Create a k -d tree with the (transformed) coordinates of the particles as input.

We construct the k -d trees in two ways: using the coordinates $(\mathbf{x}, \lambda_v \mathbf{v})$ or using the coordinates $(\mathbf{x}, \lambda_v |\mathbf{v}|)$, where λ_v is a scaling parameter. We will refer to them as the ‘full coordinate k -d tree’ and the ‘velocity norm k -d tree’, and we will denote them with a superscript $\mathbf{x}, |\mathbf{v}|$ and \mathbf{x}, \mathbf{v} , respectively. The scaling is necessary because the nearest neighbor search uses the Euclidean distance between points. There is some freedom in the choice of λ_v , which should express the ratio of a typical length divided by a typical velocity. With higher values the differences in velocity become more important than the spatial distances.

- (b) Search the nearest neighbor of each particle in the k -d tree. If the distance between particles i and j is smaller than d_{max} , merge them. Particles should not be merged multiple times during the execution of the algorithm, so mark them inactive.

We let d_{\max} be proportional to the grid spacing Δx , so particles in finer grids need to be closer to be merged. There is no single optimal way to merge two particles. Several schemes for merging are discussed below in section 3.2.

3. *Split each of the particles in **split** into two new particles.*

The new particles have the same position and velocity as the original particle i , and weights $w_i/2$ and $(w_i + 1)/2$ (both rounded down). As was discussed in section 2.2, for some applications a different method should be used.

3.2. Merge schemes

When two particles are merged, it is generally not possible to conserve both energy and momentum. Therefore we consider different schemes, that conserve either momentum, energy or other properties. The performance of these schemes is compared in section 4. We have not used ternary schemes, that merge three particles into two. As discussed in [3], such schemes do not necessarily perform better, although they can conserve both momentum and energy. Furthermore, they are more complicated to construct in 2D or 3D.

When particles i and j are merged, the weight of the new particle is always the sum of the weights $w_{\text{new}} = w_i + w_j$. For the new position we consider two choices. It can be the weighted average $\mathbf{x}_{\text{new}} = (w_i \mathbf{x}_i + w_j \mathbf{x}_j) / (w_i + w_j)$. It can also be picked randomly as either \mathbf{x}_i or \mathbf{x}_j , with the probabilities proportional to the weights. If we take the weighted average, then we introduce a (slight) bias in the spatial distribution. On the other hand, picking the position randomly increases stochastic fluctuations. For example, suppose we have a cluster of particles, and particles are being merged until there is only one left. If we use the weighted average position, then we always end up at the center of mass. So the spatial distribution of particles has become very different, a single peak at the center. With the probabilistic method we also end up with a single peak, located at the position of one of the original particles. But now the probability of ending up at particle i is proportional to w_i . Therefore, the ‘average’ spatial distribution has the same shape as before the merging.

Below we list several schemes for picking a new velocity \mathbf{v}_{new} . For convenience of notation, let

$$\begin{aligned}\mathbf{v}_{\text{avg}} &= (w_i \mathbf{v}_i + w_j \mathbf{v}_j) / (w_i + w_j), \\ v_{\text{avg}}^2 &= (w_i |\mathbf{v}_i|^2 + w_j |\mathbf{v}_j|^2) / (w_i + w_j),\end{aligned}$$

so \mathbf{v}_{avg} is the weighted average velocity and v_{avg}^2 is the weighted square norm of the velocity. The schemes are indicated by the following symbols:

p: Conserve momentum strictly by taking $\mathbf{v}_{\text{new}} = \mathbf{v}_{\text{avg}}$. Because $|\mathbf{v}_{\text{avg}}|^2 \leq v_{\text{avg}}^2$, the kinetic energy is reduced by an amount $\frac{1}{2} m w_{\text{new}} (v_{\text{avg}}^2 - |\mathbf{v}_{\text{avg}}|^2)$, where m is the mass of a particle with weight one.

ε : Conserve energy strictly by taking $\mathbf{v}_{\text{new}} = \sqrt{v_{\text{avg}}^2} \cdot \hat{\mathbf{v}}_{\text{avg}}$ (the hat denotes a unit vector). Because the energy is kept the same, the momentum increases by $m w_{\text{new}} (\sqrt{v_{\text{avg}}^2} - |\mathbf{v}_{\text{avg}}|) \cdot \hat{\mathbf{v}}_{\text{avg}}$.

\mathbf{v}_r : Conserve both momentum and energy on average, by randomly taking the velocity of one of the particles. The probability of choosing the velocity of particle i is proportional to its weight w_i .

$\mathbf{v}_r \varepsilon$: Randomly take the velocity of one of the particles, but scale it to strictly conserve energy. The expected change in momentum is $m w_{\text{new}} (\sqrt{v_{\text{avg}}^2} (w_i \hat{\mathbf{v}}_i + w_j \hat{\mathbf{v}}_j) / w_{\text{new}} - \mathbf{v}_{\text{avg}})$, which is small if $|\mathbf{v}_i| \approx |\mathbf{v}_j|$.

Although they are quite simple, we are not aware of other authors that have used schemes with randomness. It is possible to use multiple schemes, where the choice of scheme depends on the properties of the particles to be merged.

4. Numerical tests and results

It is difficult to come up with a general test of the performance of an APM algorithm. The algorithm should not significantly alter the simulation results, compared to a run without super-particles. At the same time, it should decrease the computational cost as much as possible. But whether these criteria are met depends on the particular simulation that is performed. Therefore we perform tests on a simplified system, and we focus on the effects of the coalescence algorithm on the particle distribution.

As stated before, our method works in 1D, 2D, 3D or any other dimension. For testing, we use a 2D domain with periodic boundary conditions. The domain consists of 2×2 cells, each of size 1×1 . (We let lengths and velocities be of order unity, and give them without a unit.) Initially, particles with weight 1 are distributed uniformly over the domain. Then the coalescence algorithm is performed once, with the desired weight of the particles set to 2. We compare how the different merge schemes change the momentum and energy distribution. We also measure their effect on the density, momentum and energy grid moments.

4.1. Effect of the merge schemes on the energy and momentum distribution

4.1.1. First test

In the first test, there are 400 particles with a Gaussian velocity distribution. Both components of the velocity have mean 1 and a standard deviation of $1/4$. The resulting energy and momentum distribution functions are shown in the top row of figure 3. We show the distribution of momentum along the first coordinate, not the total momentum of particles, therefore we label it x -momentum. To convert the velocity of a particle to momentum, we multiply it by the weight of the particle, which represents the mass.

Initially, the particles have weight 1, and a desired weight of 2. Then the particles are coalesced according to a merge scheme, and the changes in the energy, momentum and density distribution are recorded. The whole procedure is repeated 10^5 times for each scheme, using different random numbers, to reduce stochastic fluctuations. We have used both the velocity norm k -d tree (containing $\mathbf{x}, \lambda_v |\mathbf{v}|$) and the full coordinate k -d tree (containing $\mathbf{x}, \lambda_v \mathbf{v}$). Somewhat arbitrarily we took $\lambda_v = 4/5$, as the mean velocity plus the standard deviation in velocity was $5/4$.

The bottom row of figure 3 shows the effects of the merge schemes on the cumulative energy and momentum distribution function. The schemes are indicated by the same symbols as in section 3.2:

- p: conserve momentum
- ε : conserve energy
- \mathbf{v}_r : take velocity of one of the particles at random
- $\mathbf{v}_r \varepsilon$: take velocity from one of the particles at random, scale to conserve energy
- $\mathbf{x}, |\mathbf{v}|$: velocity norm k -d tree
- \mathbf{x}, \mathbf{v} : full coordinate k -d tree

We present the cumulative differences because they are less noisy and reveal trends more clearly. The schemes $\varepsilon^{\mathbf{x}, |\mathbf{v}|}$ and $\mathbf{v}_r \varepsilon^{\mathbf{x}, |\mathbf{v}|}$ have the same effect on the energy distribution, so they are shown together there as $(\mathbf{v}_r) \varepsilon^{\mathbf{x}, |\mathbf{v}|}$. The schemes $\mathbf{v}_r^{\mathbf{x}, |\mathbf{v}|}$ and $\mathbf{v}_r^{\mathbf{x}, \mathbf{v}}$ are also shown together, as \mathbf{v}_r . They take the new velocity randomly from one of the original particles. Therefore, on average, both do not change the shape of the energy and momentum distribution. The other schemes move particles from the tails of the distribution towards the center. To see this in the cumulative distribution functions, note that particles get removed where the slope is negative, and are moved to where the slope is positive. This happens because these schemes take averages, which are more likely to lie towards the center of the distribution. Results are not shown for the velocity norm k -d tree with the momentum conserving scheme, $\mathbf{p}^{\mathbf{x}, |\mathbf{v}|}$. This combination leads to large changes in the energy distribution.

For all the merge schemes, on average about 40% of the particles is merged. The number is below 50% because the k -d tree is created only once, in a static way. When a particle is merged, it is not removed from the tree, but marked as inactive. So it might later be the nearest neighbor of another particle, that is still to be merged. In that case, the second particle is not merged, and the algorithm moves on to the next particle. Another option would be to search for the second closest neighbor, and so on. But then merging would happen over greater distances towards the end of the algorithm.

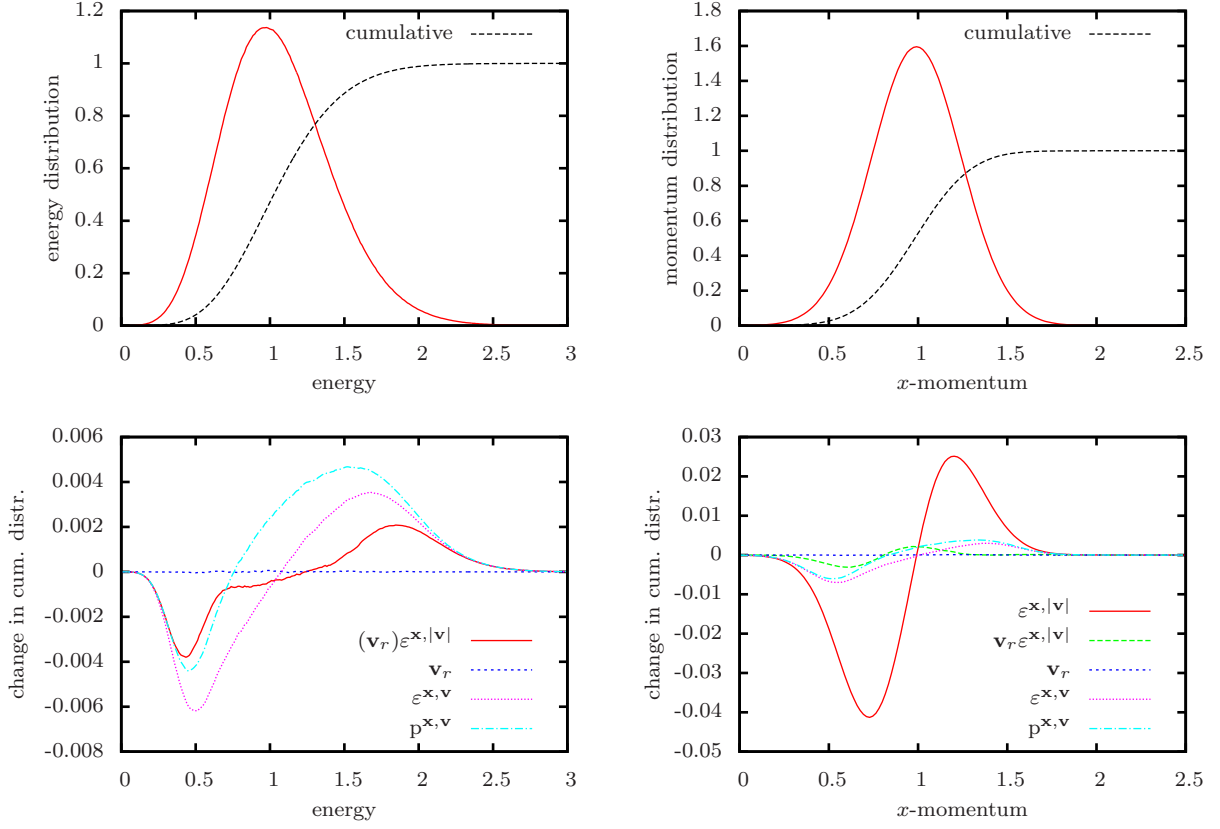


Figure 3: Results for the first test. Top row: the initial energy (left) and momentum (right) distribution of the particles. The integrated or cumulative curves are also shown (dashed). Bottom row: the effect of various merge schemes on the cumulative energy (left) and momentum (right) distribution function. The schemes are indicated by the following symbols; ε : conserve energy, \mathbf{p} : conserve momentum, \mathbf{v}_r : conserve energy and momentum on average, $\mathbf{v}_r\varepsilon$: take velocity from one of the particles at random, scale to conserve energy, $\mathbf{x},|\mathbf{v}|$: velocity norm k -d tree, \mathbf{x},\mathbf{v} : full coordinate k -d tree.

4.1.2. Second test

The second test is performed in the same way as the first test, but now the particles have a different velocity distribution. Both components of the velocity have a mean of $1/4$ and a standard deviation of 1. The resulting energy and momentum distribution functions are shown in the top row of figure 4. Because it is more isotropic, the second velocity distribution poses a bigger challenge for the merge schemes. The bottom row of figure 4 shows the effects of the merge schemes on the cumulative energy and momentum distribution function. Again, the schemes \mathbf{v}_r perform best, as the other schemes move particles from the tail of the distribution towards the center. Note that the schemes $\mathbf{v}_r\varepsilon^{\mathbf{x},|\mathbf{v}|}$ and $\varepsilon^{\mathbf{x},\mathbf{v}}$ also move particles away from zero momentum. As for the first test, on average about 40% of the particles is merged.

4.2. Effect on grid moments

In many particle simulations, a grid (or mesh) is used. Grid moments are defined at the grid points, and provide local averages from which the fields acting on the particles can be computed. For example, the first grid moment gives the particle density, the second the current density or momentum density, the third the energy density and so on. Particles can be mapped to grid moments in different ways, here we use first order interpolation, also known as cloud-in-cell (CIC) [1, 2].

Using the data of the second test, we now look at the effect of the merge schemes on the first three grid moments. An APM algorithm should not induce large differences in these grid moments. The mean

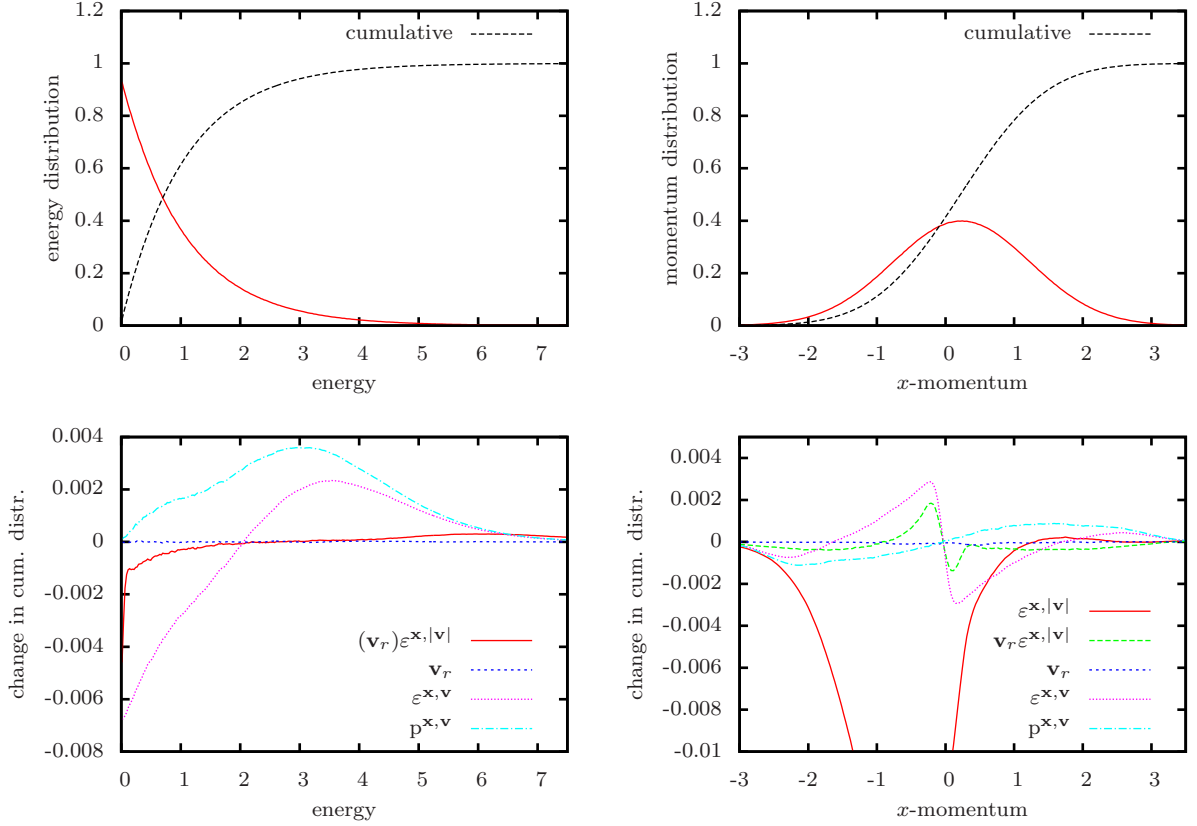


Figure 4: Results for the second test. Top row: the initial energy (left) and momentum (right) distribution of the particles. The integrated or cumulative curves are also shown (dashed). Bottom row: the effect of various merge schemes on the cumulative energy (left) and momentum (right) distribution function. The legend is the same as for figure 3. In the right figure, the peak for scheme 3ε is cut off, it extends to -0.018 .

difference is often zero, because the corresponding quantity is conserved. Therefore, we also look at the relative standard deviation, or σ/μ , where σ is the standard deviation of a random variable with mean μ . This is a measure of the relative size of fluctuations. We measure these fluctuations at a single grid point, as they would be correlated for multiple grid points. In table 1 the changes in the grid moments are given for various schemes. The schemes are labeled by the same symbols as before. In addition, \mathbf{x}_r indicates that the new position is picked randomly from one of the merged particles. The bottom part of the table is about cell-by-cell merging, which is discussed in section 4.3. The APM fluctuations should be compared to those resulting from advancing the particles in time. Therefore, the table includes entries that list the effect of taking a timestep Δt without any merging. Since we have included no collisions, the particles simply move with a constant velocity during this timestep.

The average deviation in particle density ρ is zero for all the schemes, because they conserve the total weight of the particles. Therefore this quantity is not included in table 1. The induced fluctuations in the grid moments can differ by almost an order of magnitude between the schemes. As expected, conserving momentum reduces the mean energy, and conserving energy increases momentum. This is especially problematic when the velocity norm k -d tree is used. The mean deviations are then larger than 10%. The full coordinate k -d tree in combination with the energy-conserving scheme, $\varepsilon^{\mathbf{x},\mathbf{v}}$, gives good results regarding energy and momentum conservation. Schemes that select the new velocity at random do not lead to systematic differences in the energy and momentum grid moments. With the $\mathbf{v}_r^{\mathbf{x},|\mathbf{v}|}$ scheme, the fluctuations in momentum can be relatively large. The $\mathbf{v}_r^{\mathbf{x},\mathbf{v}}$ scheme leads to much smaller fluctuations. This scheme

Method	N_{merge}	d_{avg}	σ_ρ	Δp_x	σ_{p_x}	$\Delta \varepsilon$	σ_ε
$\Delta t = 0.1$	-	-	1.6%	0.0%	9%	0.0%	3.8%
$\Delta t = 0.2$	-	-	2.9%	0.0%	16%	0.0%	6.7%
$\Delta t = 0.4$	-	-	4.9%	0.0%	25%	0.0%	9.4%
$\varepsilon^{\mathbf{x}, \mathbf{v} }$	39%	0.16	0.3%	12%	16%	0.0%	0.8%
$\mathbf{p}^{\mathbf{x}, \mathbf{v} }$	39%	0.16	0.3%	0.0%	4%	-37%	5.0%
$\mathbf{v}_r^{\mathbf{x}, \mathbf{v} }$	39%	0.16	0.3%	0.0%	24%	0.0%	1.2%
$\mathbf{v}_r \varepsilon^{\mathbf{x}, \mathbf{v} }$	39%	0.16	0.3%	0.4%	25%	0.0%	0.8%
$\mathbf{v}_r \mathbf{x}_r^{\mathbf{x}, \mathbf{v} }$	39%	0.16	1.0%	0.0%	24%	0.0%	2.2%
$\varepsilon^{\mathbf{x},\mathbf{v}}$	40%	0.38	0.7%	0.1%	4%	0.0%	1.5%
$\mathbf{p}^{\mathbf{x},\mathbf{v}}$	40%	0.38	0.7%	0.0%	4%	-1.2%	1.5%
$\mathbf{v}_r^{\mathbf{x},\mathbf{v}}$	40%	0.38	0.7%	0.0%	6%	0.0%	2.4%
$\mathbf{p}^{\mathbf{v}}, \text{cell}$	40%	0.19	2.8%	0.0%	12%	-0.9%	3.8%
$\mathbf{v}_r^{\mathbf{x}, \mathbf{v} }, \text{cell}$	39%	0.17	0.3%	0.0%	23%	0.0%	1.5%
$\mathbf{v}_r \mathbf{x}_r^{\mathbf{x}, \mathbf{v} }, \text{cell}$	39%	0.17	1.0%	0.0%	23%	0.0%	2.2%
$\varepsilon^{\mathbf{x},\mathbf{v}}, \text{cell}$	38%	0.40	0.8%	0.5%	5%	0.0%	1.8%

Table 1: The induced differences and fluctuations in the grid moments by the various merge schemes, using the second test distribution. Legend: N_{merge} is the fraction of merged particles, and d_{avg} is the average distance between merged particles. The relative differences in grid moments are indicated by Δp_x (momentum) and $\Delta \varepsilon$ (energy), and relative standard deviations by σ_ρ (density), σ_{p_x} (momentum) and σ_ε (energy). Both are given relative to the mean value. The rows starting with Δt show the fluctuations in the grid moments resulting from a timestep (no merging). The merge schemes are indicated by the following symbols; ε : conserve energy, \mathbf{p} : conserve momentum, \mathbf{v}_r : random velocity, $\mathbf{v}_r \varepsilon$: random velocity, scale to conserve energy, \mathbf{x}_r : random position, \mathbf{v} : k -d tree contains only the velocity, $\mathbf{x},|\mathbf{v}|$: velocity norm k -d tree, \mathbf{x},\mathbf{v} : full coordinate k -d tree, cell: perform the merging cell-by-cell.

performs well: on average it conserves the grid moments and also the shapes of the energy/momentum distribution functions, and it does not create big fluctuations.

Taking the new position at random at one of the original particles (\mathbf{x}_r) increases the fluctuations in particle density. For all the schemes, the fluctuations in density, momentum or energy are smaller than those resulting from a timestep of $\Delta t = 0.4$.

4.3. Cell-by-cell merging

Using k -d trees, there is no reason to do cell-by-cell merging. But because this type of merging is commonly used, we briefly evaluate its effects. The bottom part of table 1 shows results for cell-by-cell merging, for various schemes, using the second test distribution. The notation is the same as before, and a superscript \mathbf{v} indicates that only the velocity was used in the k -d tree, not the position. The fluctuations are mostly similar if the particles are merged locally (cell-by-cell) instead of globally. With fewer particles per cell, the differences would be larger though, as close neighbors are more likely to lie in other cells.

The average spatial distribution of particles directly after merging are shown in figure 5. Only the type of k -d tree is important for the effect, because all the shown merge schemes take the average position. From left to right: With the velocity norm k -d tree the spatial distribution of particles is affected close to the cell boundaries. With the full coordinate k -d tree, the effect is similar as with the velocity norm k -d tree. Using the k -d tree that includes only the velocity, particles are moved to the center of the cells. The spatial distribution is severely affected. Furthermore, the fluctuations in particle density are higher, as can be seen in table 1. If particles would be merged globally (not cell-by-cell), the spatial distribution would be uniform.

4.4. Computational costs of k -d trees

The goal of an APM algorithm is to speed up a simulation, so the algorithm itself should not take too much time. Theoretically, the computational complexity of creating a k -d tree is $O(N_p \log N_p)$, with N_p the number of points in the tree. The average cost of a random search in the tree is $O(\log N_p)$. We have tested the practical performance of the KDTree2 library on an Intel i7-2600 CPU. In figure 6 the creation time and the average search time are shown for k -d trees of various sizes. Neighbors can be found faster if

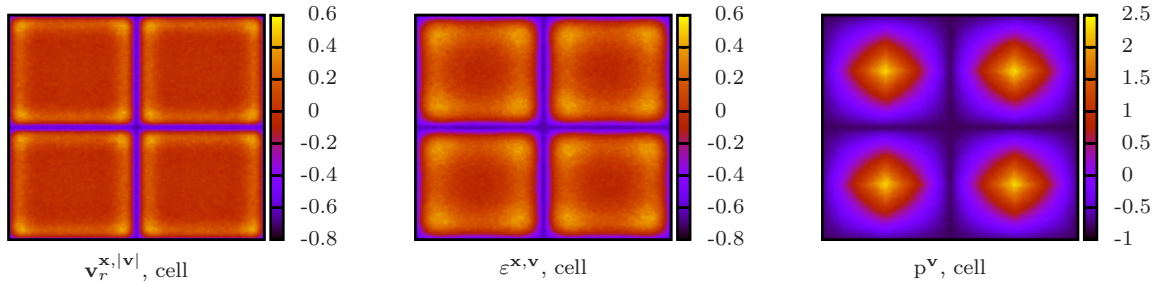


Figure 5: Results for cell-by-cell merging. Shown are the relative differences in particle density, for different ways of creating the k -d tree. The schemes are indicated by the same symbols as in table 1 and figure 3. Cell-by-cell merging leads to clear artifacts, especially when the k -d tree contains only the velocity.

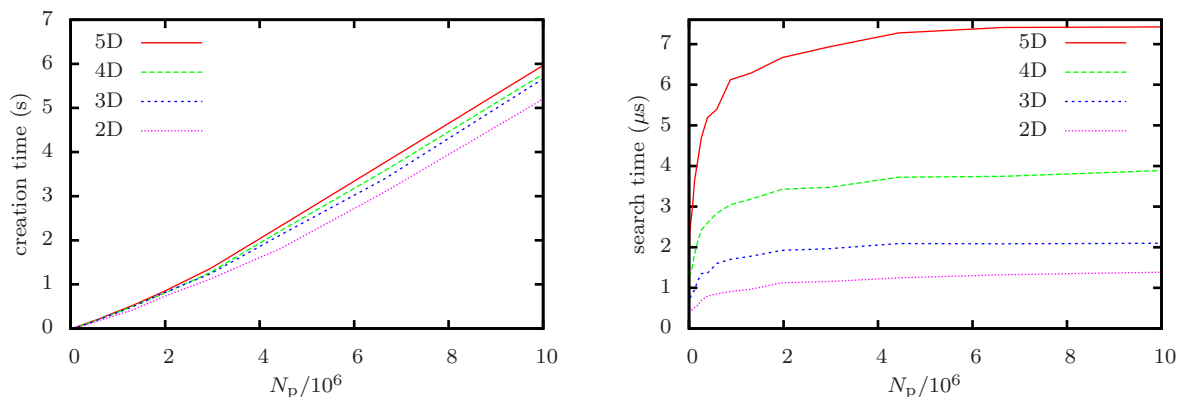


Figure 6: Performance figures for k -d trees in 2D-5D with N_p points, using the KDTree2 [14] library. Left: the time it takes to create the k -d tree. Right: the time it takes to find a nearest neighbor (for uncorrelated searches). The calculations were performed on an Intel i7-2600 CPU.

the k -d tree is constructed in fewer dimensions. Note that the average search time is given for uncorrelated searches, that are done at random locations. This is the worst-case scenario, as the CPU cannot do efficient data caching. If the next search location is picked close to the previous search location, search times in 5D decrease by more than 80%.

5. Conclusion

Adaptively adjusting the weights of simulated particles can greatly improve the efficiency of simulations. We follow Welch et al. [7] and call algorithms that do this ‘adaptive particle management’ (APM) algorithms. In this work, we have focused on the pairwise merging of particles. We found that the use of a k -d tree offers several important advantages over present methods. First, only particles that are ‘close together’ are merged. ‘Close together’ can be defined as desired (for example close in position and velocity). This ensures that the distribution of particles is not significantly altered. Second, the merging can be performed completely independent of the numerical mesh used in the simulation. The algorithm works in the same way, whether the simulation is in 1D or in any higher dimension. Third, with a k -d tree, the closest neighbors can be located efficiently. Therefore, the method can be used for simulations with millions of particles. Fourth, from a practical point of view, the use of a k -d tree library greatly simplifies the implementation of pairwise merging.

Two particles can be merged in different ways, and we have compared various merge schemes. An interesting option is to select properties for the merged particle at random from the original particles. With

these stochastic schemes fluctuations increase, but on average both momentum and energy can be conserved. The optimal scheme depends on the application. In general, it is more important to preserve the essential characteristics of the particle distribution function than to exactly conserve grid moments. A scheme that conserves energy or momentum should typically be used with a full coordinate k -d tree (containing \mathbf{x}, \mathbf{v}). A velocity norm k -d tree (containing $\mathbf{x}, |\mathbf{v}|$) can be used with a stochastic scheme. The advantage of a velocity norm k -d tree is that it can be constructed and searched faster than one with the full coordinates. The combination of a stochastic scheme with a full coordinate k -d tree seems a good choice: on average, the shape of the energy and momentum distribution functions is conserved, while the induced fluctuations in the grid moments are relatively small.

Acknowledgement

J. Teunissen was supported by STW-project 10755.

References

- [1] C. K. Birdsall, A. B. Langdon, Plasma physics via computer simulation / Charles K. Birdsall, A. Bruce Langdon, McGraw-Hill, New York :, 1985.
- [2] R. W. Hockney, J. W. Eastwood, Computer simulation using particles., IOP Publishing Ltd., Bristol, England, 1988.
- [3] G. Lapenta, Particle rezoning for multidimensional kinetic particle-in-cell simulations, Journal of Computational Physics 181 (1) (2002) 317 – 337. doi:10.1006/jcph.2002.7126.
URL <http://www.sciencedirect.com/science/article/pii/S0021999102971263>
- [4] G. Lapenta, J. U. Brackbill, Dynamic and selective control of the number of particles in kinetic plasma simulations, Journal of Computational Physics 115 (1) (1994) 213 – 227. doi:10.1006/jcph.1994.1188.
URL <http://www.sciencedirect.com/science/article/pii/S0021999184711880>
- [5] G. Lapenta, J. Brackbill, Control of the number of particles in fluid and mhd particle in cell methods, Computer Physics Communications 87 (1–2) (1995) 139 – 154, particle Simulation Methods. doi:10.1016/0010-4655(94)00180-A.
URL <http://www.sciencedirect.com/science/article/pii/001046559400180A>
- [6] F. Assous, T. P. Dulimbert, J. Segré, A new method for coalescing particles in pic codes, Journal of Computational Physics 187 (2) (2003) 550 – 571. doi:10.1016/S0021-9991(03)00124-4.
URL <http://www.sciencedirect.com/science/article/pii/S0021999103001244>
- [7] D. Welch, T. Genoni, R. Clark, D. Rose, Adaptive particle management in a particle-in-cell code, Journal of Computational Physics 227 (1) (2007) 143 – 155. doi:10.1016/j.jcp.2007.07.015.
URL <http://www.sciencedirect.com/science/article/pii/S0021999107003245>
- [8] O. Chanrion, T. Neubert, A pic-mcc code for simulation of streamer propagation in air, J. Comput. Phys. 227 (15) (2008) 7222–7245. doi:10.1016/j.jcp.2008.04.016.
URL <http://dx.doi.org/10.1016/j.jcp.2008.04.016>
- [9] C. Li, U. Ebert, W. Hundsdorfer, Spatially hybrid computations for streamer discharges : Ii. fully 3d simulations, Journal of Computational Physics 231 (3) (2012) 1020 – 1050, special Issue: Computational Plasma Physics. doi:10.1016/j.jcp.2011.07.023.
URL <http://www.sciencedirect.com/science/article/pii/S0021999111004517>
- [10] G. Lapenta, Adaptive Multi-Dimensional Particle In Cell, ArXiv e-prints arXiv:0806.0830.
- [11] J. L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517. doi:10.1145/361002.361007.
URL <http://doi.acm.org/10.1145/361002.361007>
- [12] G. D. Moss, V. P. Pasko, N. Liu, G. Veronis, Monte carlo model for analysis of thermal runaway electrons in streamer tips in transient luminous events and streamer zones of lightning leaders, Journal of Geophysical Research 111 (2006) A02307 – A02344. doi:10.1029/2005JA011350.
- [13] C. Shon, H. Lee, J. Lee, Method to increase the simulation speed of particle-in-cell (pic) code, Computer Physics Communications 141 (3) (2001) 322 – 329. doi:10.1016/S0010-4655(01)00417-9.
URL <http://www.sciencedirect.com/science/article/pii/S0010465501004179>
- [14] M. B. Kennel, KDTree 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space, ArXiv Physics e-prints arXiv:arXiv:physics/0408067.